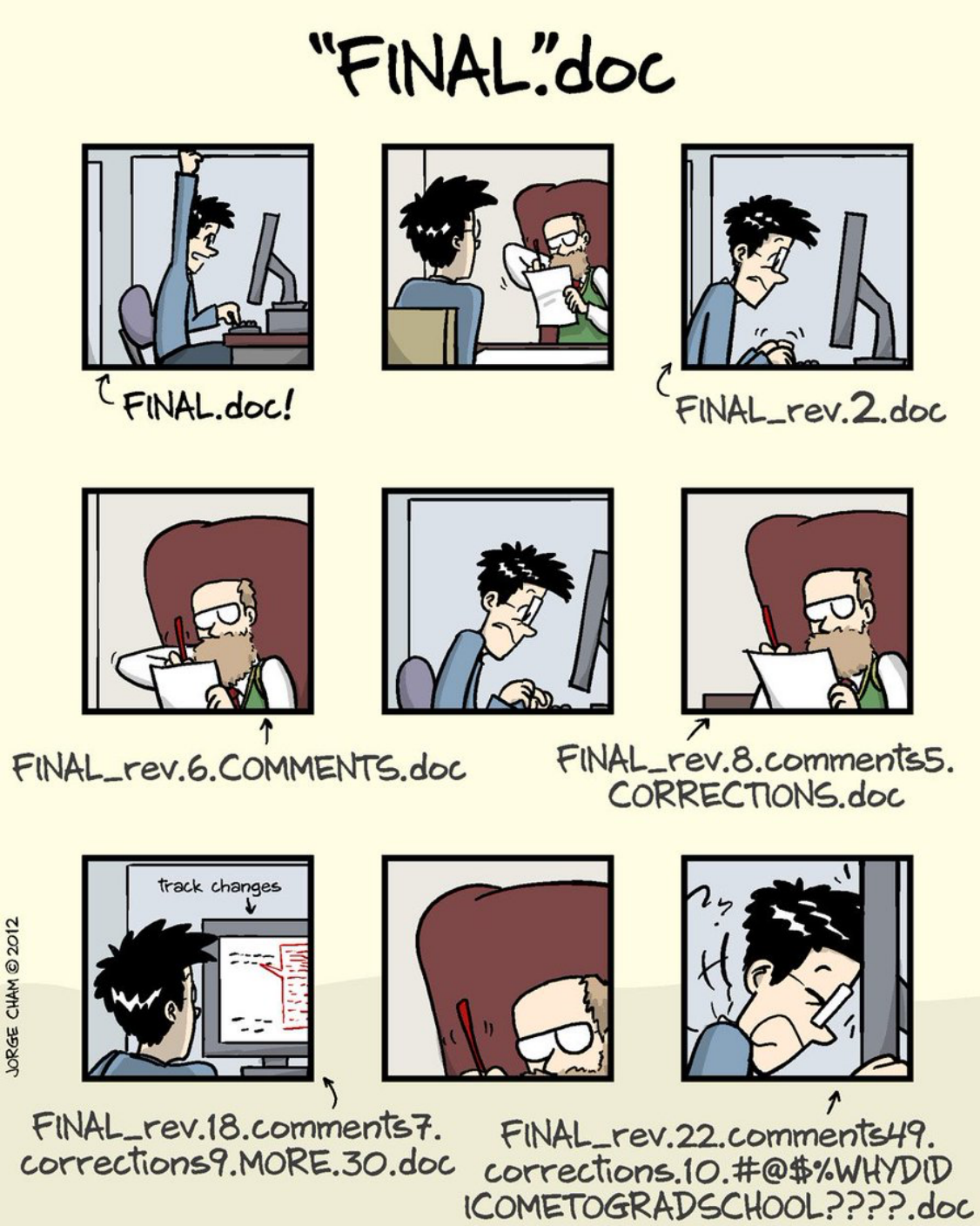


WHAT IS VERSION CONTROL AND WHY USE IT?

VERSION CONTROL

- A system that manages and keeps records of changes made to files
- Allows for collaborative development
- Allows you to know who made what changes and when
- **Allows you to revert any changes and go back to a previous state**

Article

Excuse Me, Do You Have a Moment to Talk About Version Control?

Jennifer Bryan

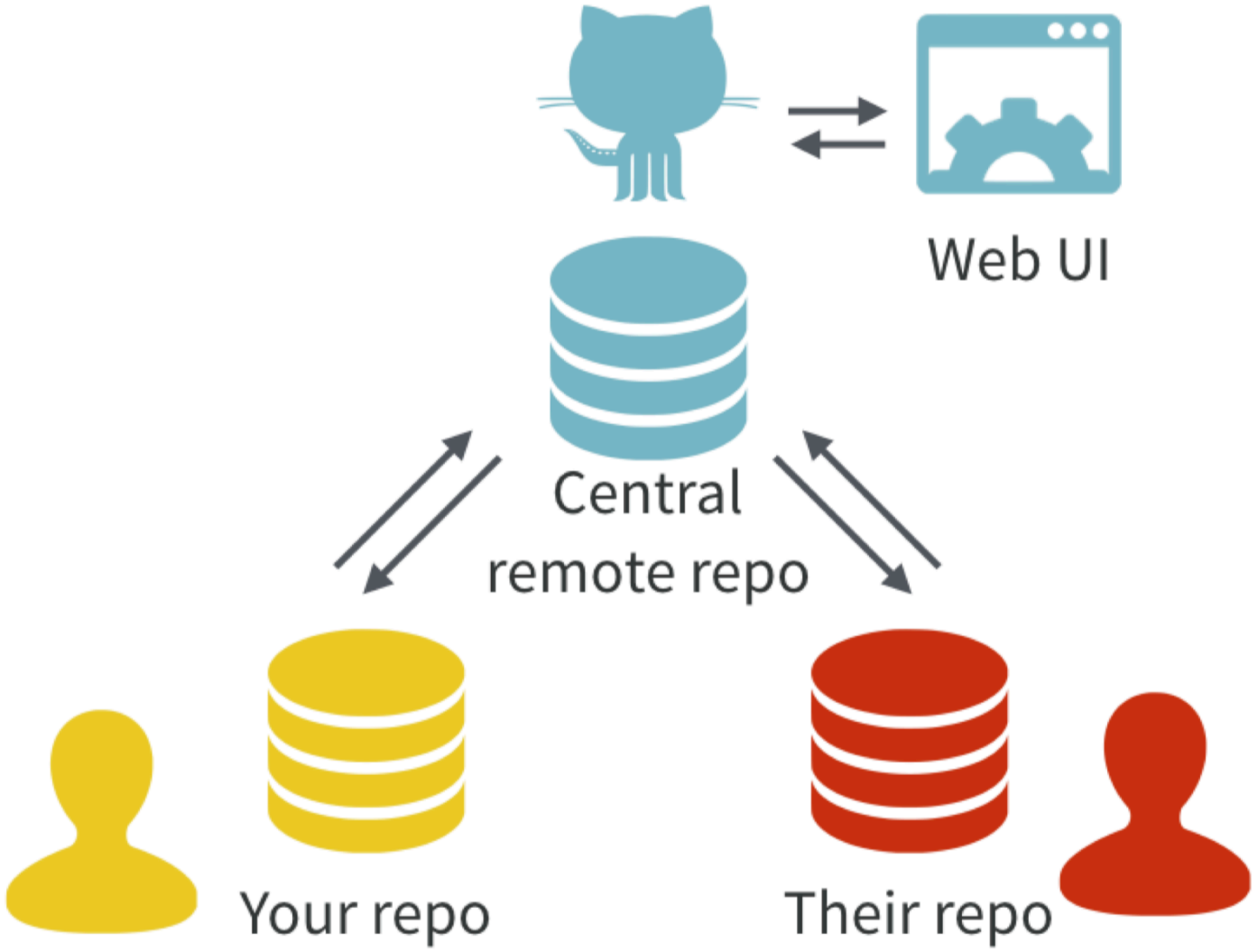
Pages 20-27 | Received 01 Jul 2017, Accepted author version posted online: 14 Nov 2017, Published online: 24 Apr 2018

Download citation https://doi.org/10.1080/00031305.2017.1399928 Check for updates

WHAT ARE GIT AND GITHUB?

- Git and GitHub are **two different things**
- Git is a particular implementation of version control originally designed by Linus Torvalds in 2005 as a way of managing the Linux kernel. Git manages the evolution of a set of files – called a repository or repo
- Essentially, the language of version control

- GitHub is an online hub for hosting Git repositories and provides GUI software for using Git
- GitHub complements Git by providing a slick user interface and distribution mechanism for repositories. Git is the software you will use locally to record changes to a set of files. GitHub is a hosting service that provides a Git-aware home for such projects on the internet
- GitHub is like DropBox or Google Drive, but more structured, powerful, and programmatic



With Git, all contributors have a copy of the repo, with all files and the full history. It is typical to stay in sync through the use of a central remote repo, such as GitHub. Hosted remotes like GitHub also provide access to the repo through a web browser.

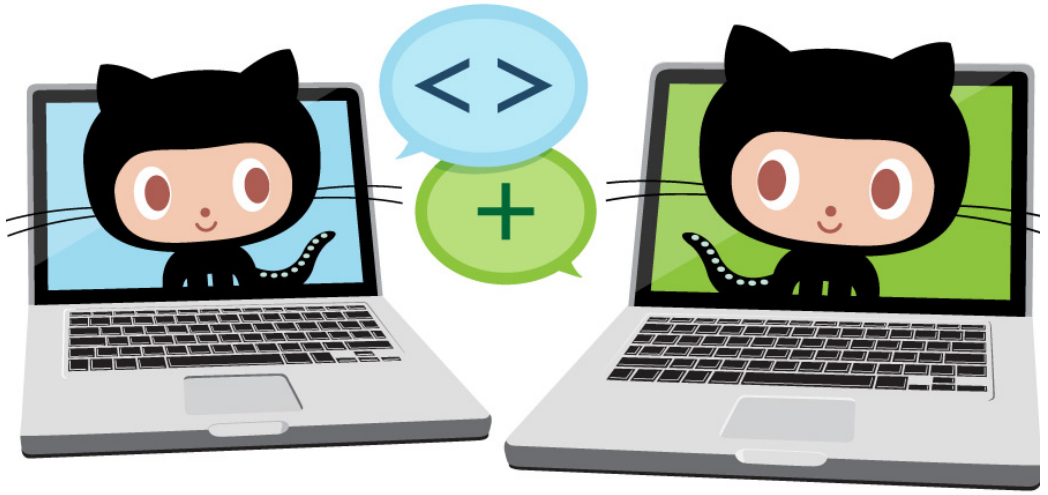
HOW DOES GIT WORK: KEY CONCEPTS

- Git keeps track of your files using “snapshots” that record what the files in your repository look like at any given point in time
- You decide when to take a snapshot and of what files, this is known as a **commit**
    - Can be used as noun or verb
    - *“I commited code”*
      “I just made a new commit”
- Have the ability to go back and visit any commit
- A project is made up of a bunch of commits

KEY CONCEPTS: COMMIT

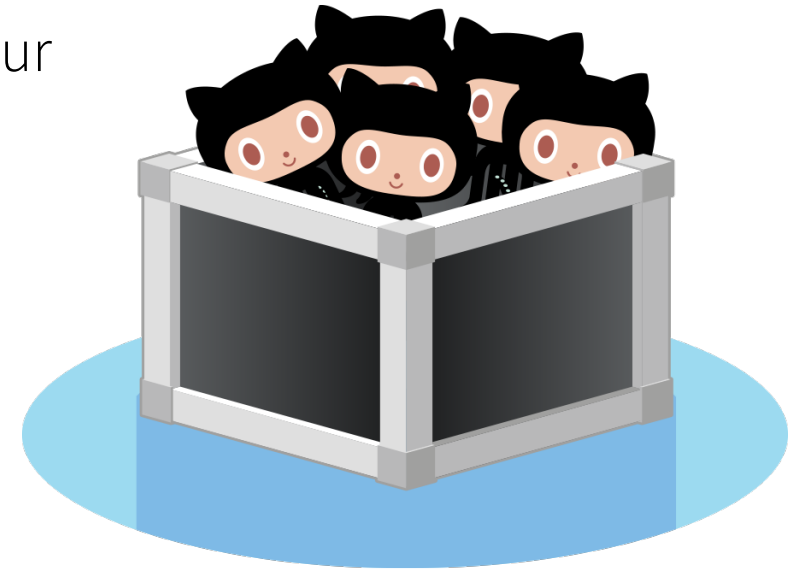
Commits contain 3 pieces of information:

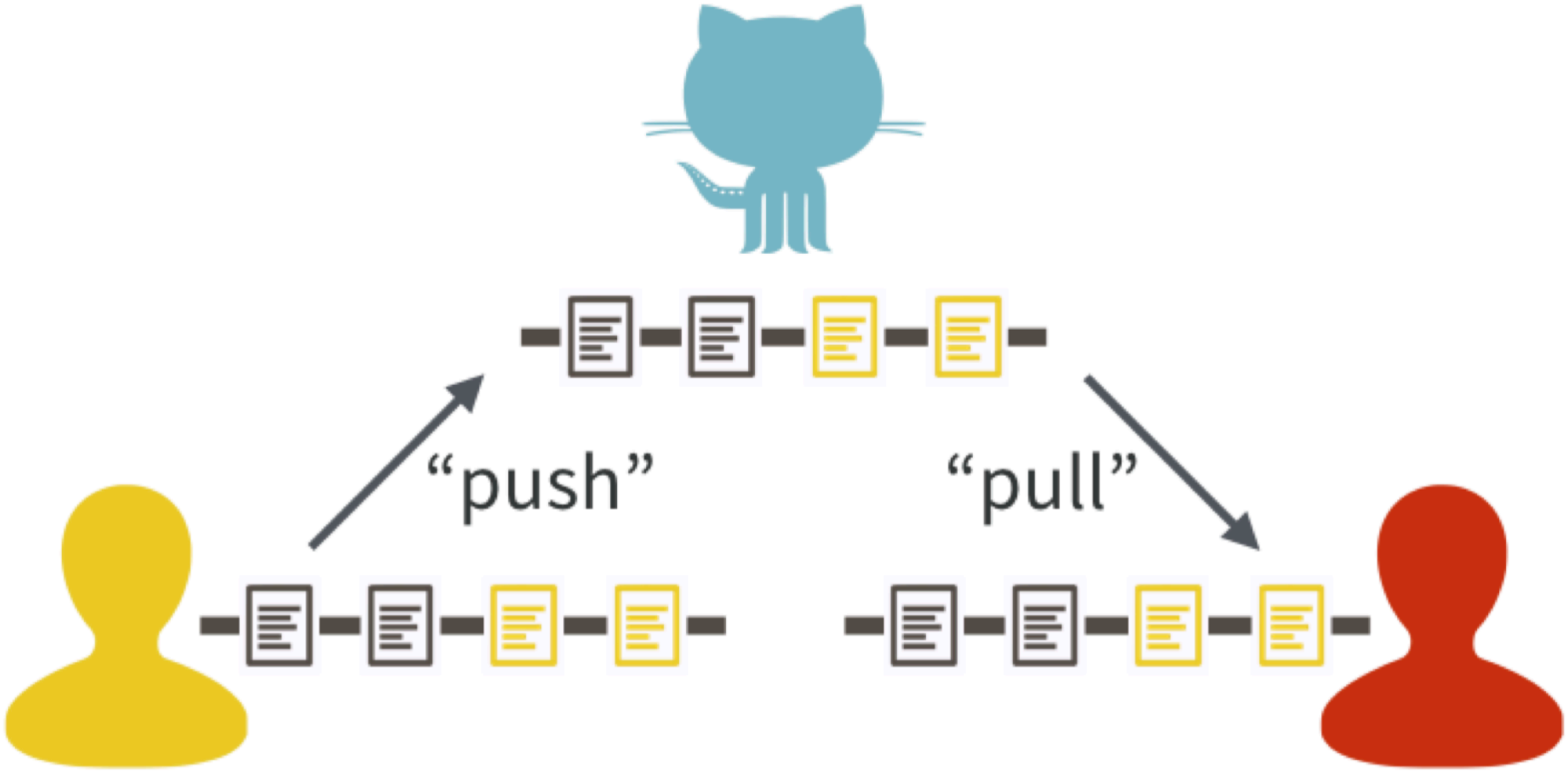
1. Information about how the files have changed from the previous commit
2. A reference to the commit that came before it, known as the **parent commit**
3. An SHA-1 hash code (e.g. fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e)



KEY CONCEPTS: REPOSITORIES

- Often referred to as a **repo**, this a collection of all your files and the history of those files (i.e. commits)
- Can live on a local machine or on a remote server (e.g. GitHub)
- The act of copying a repo from a remote server is called cloning
- Cloning from a remote server allows teams to work together
- The process of downloading commits that don't exist on your machine from a remote server is called **pulling** changes
- The process of adding your local changes to a remote repo is called **pushing** changes



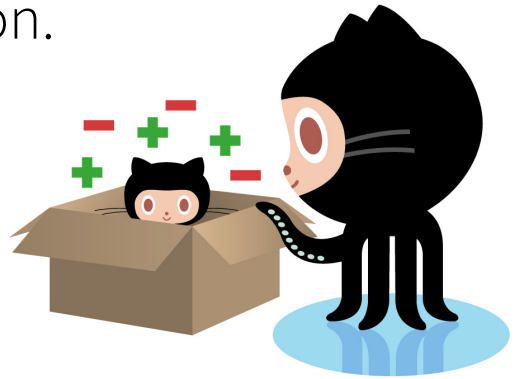


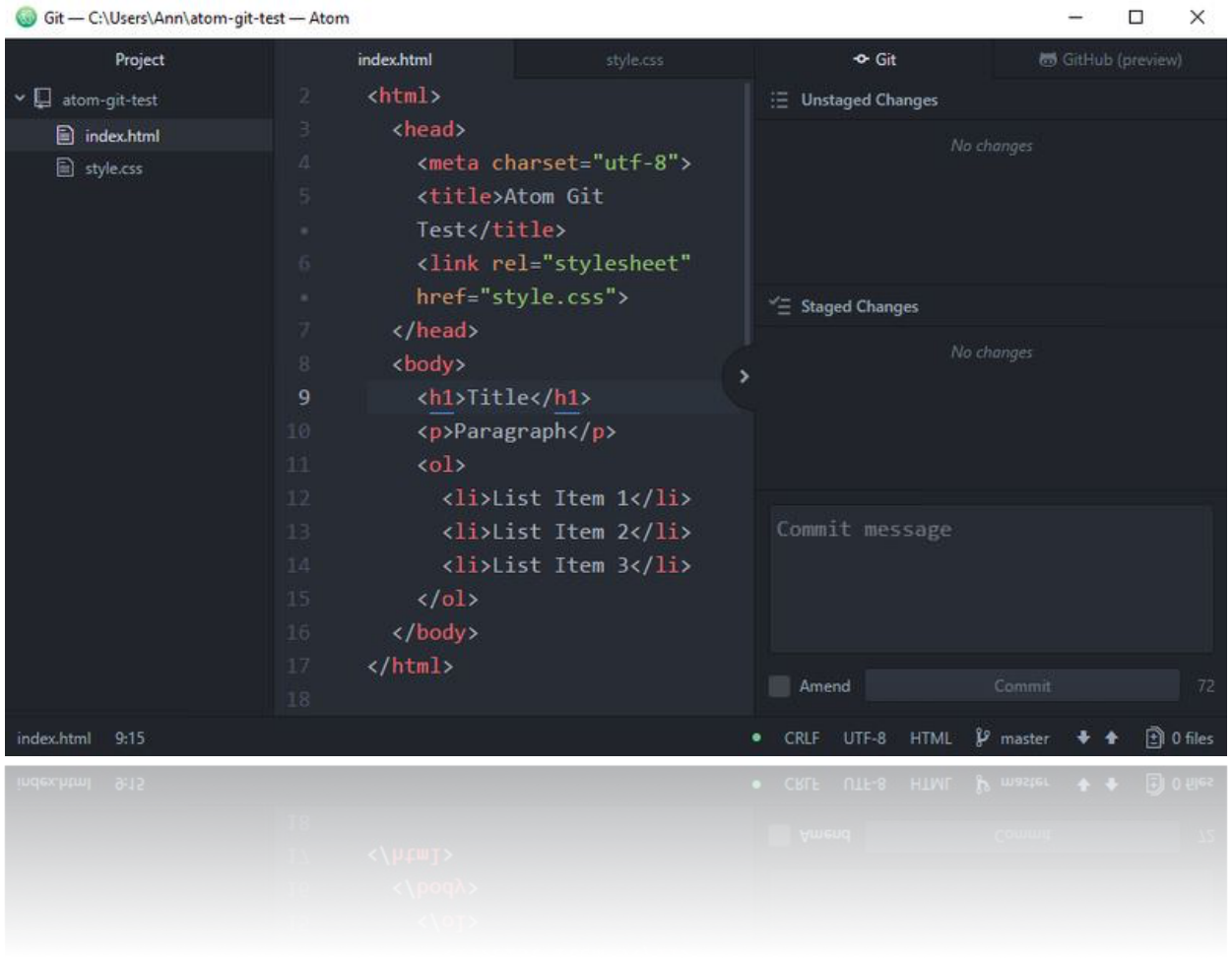
One contributor has made two new commits and updates the master copy on GitHub with a push.
Another contributor stays up-to-date with a pull from GitHub.

KEY CONCEPS: MAKING A COMMIT

The process:

- Make some changes to a file
- Use the `git add` command or Atom to add files to the **staging area**
- Type a commit message (that shortly describes the changes you made since the last commit) into the commit message box, and click the Commit button.





WRITING A GOOD COMMIT MESSAGE

The 7 rules of a great commit message:

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and *why* vs. *how*

WRITING A GOOD COMMIT MESSAGE

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Further reading:
chris.beams.io/posts/git-commit/